(2)

CECOM

CENTER FOR SOFTWARE ENGINEERING

ADVANCED SOFTWARE TECHNOLOGY

Subject: Final Report - Establish and Evaluate Ada Runtime Features of Interest for Real-Time Systems

CIN: C02 092LA 0003

15 FEBRUARY 1990

FINAL REPORT
ESTABLISH AND EVALUATE
ADA RUNTIME FEATURES OF
INTEREST FOR REAL-TIME SYSTEMS

CONTRACT NUMBER: MDA 903-87-D-0056

IITRI PROJECT NUMBER: T06168

PREPARED FOR:

U.S. ARMY, CECOM
ADVANCED SOFTWARE TECHNOLOGY
AMSEL-RD-SE-AST-SS-R
FT. MONMOUTH, NJ 07703-5000

PREPARED BY:

IIT RESEARCH INSTITUTE
4600 FORBES BLVD.
LANHAM, MD 20706

DECEMBER 1988

EXECUTIVE SUMMARY

The objective of this study was to provide software developers with guidance in the selection of Runtime Environments (RTEs) to ensure that all timing and storage requirements of real-time embedded systems can be met. Because there is no "universal best' runtime environment (RTE), the selection of an RTE is domain specific. This study developed a step-by-step process that a developer can use to evaluate RTEs. This process was applied to one class of systems, Communication and Electronic Intelligence (COMINT/ELINT) systems.

A process was developed to determine which Ada runtime features were important for real-time embedded systems. This process involved prioritizing Ada RTE elements by the implementation of a prioritization matrix. The prioritization matrix was demonstrated by prioritizing RTE elements for COMINT/ELINT systems. The prioritization matrix was designed so it could be applied to any class of real-time embedded systems with only slight modifications.

The prioritized RTE elements were used to prioritize groups of benchmarks. This provided software developers with a prioritized list of groups of benchmarks that measure the critical areas of candidate RTEs being considered for COMINT/ELINT systems.

The concept of a composite benchmark was developed as another means to test candidate RTEs. Unlike most existing benchmarks, a composite benchmark takes into account the interactions and interfaces that go on within a system. A preliminary composite benchmark description was developed for COMINT/ELINT systems.

When selecting an RTE, the composite benchmark would be used to test the minimum threshold of RTEs. Then, the prioritized groups of benchmarks would be used to test the critical RTE elements to determine which RTEs perform best in the critical areas.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

1.1 BACKGROUND

Since the early implementation of the Ada language, Ada compilers were required to pass a validation test. Thus, the primary goal of compiler vendors was to have their compiler pass this validation test. This left performance as a secondary issue. In addition, the Department of Defense (DOD) mandates the use of Ada in the development of real-time embedded systems. With 206 validated compilers (and the list continues to grow), software developers must be able to obtain guidance in the selection of a compiler and its runtime environment (RTE) to ensure all the strict timing and storage requirements of real-time embedded systems are met.

To provide this guidance requires the identification of Ada RTE features of interest for real-time embedded systems. This involves two steps: first, the Ada runtime features that are important for real-time systems need to be established; second, evaluation criteria to evaluate these features need to be determined. To establish the important Ada runtime features, the Ada RTE elements need to be prioritized. To evaluate the Ada runtime features, current benchmarks must be prioritized and new benchmarks developed.

1.2 PURPOSES OF THIS STUDY

The primary purpose of this study was to assist software developers in selecting an RTE that meets the performance requirements of their application. This study provides a process that a developer can use to prioritize RTE elements for a particular application domain. The prioritization of RTE elements provides the means to prioritize benchmarks. Since it might not be possible for a developer to run all possible benchmarks, a listing of prioritized groups of benchmarks allows developers to focus on the most critical areas of a specific application domain.

In this study RTE elements were prioritized for one class of systems supported by U.S. Army Communications-Electronics Command (CECOM) Ft. Monmouth, NJ. That class is Communication and Electronic Intelligence (COMINT/ELINT) systems.

Most existing benchmarks test one RTE element in isolation, without consideration of the effect of RTE elements interfacing. To measure both the specific elements of interest in an RTE, as well as any interaction effects, required a new type of benchmark.

One major result of this study was the formulation of a composite benchmark that tests RTE elements and their interactions. A composite benchmark is defined as a prototype of the capabilities of a particular class of system. The goal of a composite benchmark is to test the minimum threshold of all candidate RTEs. This study developed preliminary guidelines for developing a composite benchmark and a preliminary description for a composite benchmark for COMINT/ELINT systems.

1.3 ORGANIZATION OF THIS REPORT

The organization of this report reflects the order in which the research was done. In the first step (Section 2.0) a class of systems, COMINT/ELINT, were chosen to be studied and the capabilities common to COMINT/ELINT systems were identified. The second step (Section 3.0) identified what Ada constructs would be used to implement each capability. The third step (Section 4.0) identified which RTE element would be used to support the implementation of each identified Ada construct. The forth step (Section 5.0) prioritized the Ada RTE elements by applying a prioritization matrix to the RTE elements. The fifth step (Section 6.0) used the prioritized RTE elements to prioritize groups of benchmarks. In the sixth step (Section 7.0) the purpose of a composite benchmark, the development process of a composite benchmark, and a preliminary description of a composite benchmark are given. The last step (Section 8.0) recommends how benchmarks are to be used in the RTE selection process.

2.0 SELECTION PROCESS OVERVIEW

The first step in this research was to select specific real-time systems to study. The goal of the selection process was to identify a class of systems supported by CECOM that are the most challenging to develop and maintain and then to select representative systems from that class. The most challenging class of systems was the class with the greatest number of large, real-time systems. Once the class was identified, sample systems were chosen. It would have been impractical to study all of the systems because of the system diversity and the number of systems.

At the time the research was performed for this report, CECOM supported 140 systems. CECOM classified their systems into five Battlefield Functional Areas (BFA), and each BFA was divided into its own set of categories. Individual systems were placed into categories within each BFA. Figure 2-1 shows a diagram of the classification scheme.

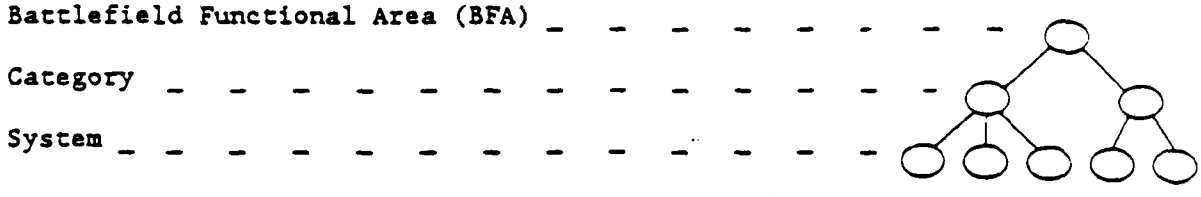

Figure 2-1. The Classification Scheme.

CECOM supports the following BFAs.

1. Intelligence Electronic Warfare (IEW)
2. Fire Support (FS)
3. Maneuver Control (MC)
4. Communications (COMM)
5. Air Defense (AD).

The initial data set contained information on 136 systems; however, only 56 of the systems have enough information on them to adequately analyze. Therefore, the initial analysis was done on these 56 systems.

The selection process involved three stages. First, the real-time systems in the data set were identified; second, the class of systems with the greatest number of large real-time systems was determined; and third, specific systems within the chosen class were selected.

2.1 REAL-TIME SYSTEMS IDENTIFICATION

Real-time software constantly monitors, analyzes, and responds to external physical events in a time-critical fashion. The high-level functions performed by real-time systems are as follows:

1. Monitor - connection of a physical event to a computer system so that data pertaining to the physical event can be collected.
2. Analyze - portion of software that determines the next course of action based on the data collected.
3. Respond - portion of software that executes the course of action determined from the analysis.

Real-time systems perform one or more of these high-level functions and must also be time critical in that the failure to monitor, analyze, and respond in a timely manner would be disastrous.

To determine which CECOM systems were real-time systems, the functions that each system performs were compared to the three high-level functions listed above. If CECOM systems implemented one or more of these high-level functions and the function were time-critical, the system was deemed to be a real-time system.

TABLE 2-1 classifies the functions performed by CECOM systems according to the high-level functions they implement. All the functions were determined to be time-critical. A description of each function is found in Appendix A.

TABLE 2-1

System Functions By the High-Level Functions They Implement

Monitor	Analyze	Respond
receiver	direction finding	transmitter
reception	analysis	countermeasure
target detection	location	antenna controller

2.2 CLASS SELECTION

After determining what CECOM systems were real-time systems, the next step was to pick a particular class of CECOM systems to study. The objective was to choose a class of systems that had the largest number of large real-time systems. Because CECOM already classified systems into BFAs and categories, the objective was to choose a BFA and category that contained the largest number of real-time systems. TABLE 2-2 shows the number of real-time systems in each BFA.

TABLE 2-2

Number of Real-Time Systems in Each BFA

BFA	Number of Real-Time Systems
IEW	14
FS	3
MC	1
COMM	5
AD	5

IEW was chosen as the BFA to be studied because it contained the most large real-time systems. The IEW category of Communication Intelligence and Electronic Intelligence (COMINT/ELINT) contained more real-time systems than any other IEW category; therefore, it was selected to be studied.

2.3 SYSTEM SELECTION

The last stage was to select specific systems to study that were representative of COMINT/ELINT systems. An assumption was made that if a method could be developed to prioritize RTE elements for large systems, the method could be applied to smaller types of real-time systems.

Four COMINT/ELINT systems were selected to represent all COMINT/ELINT systems: Improved Guardrail V (IGRV), Advanced Quicklook (AQL), Communication High Accuracy Airborne Location System (CHAALS), and Trailblazer B. Three of these systems are part of the Guardrail Common Sensor Family, which contains the largest systems within IEW.

IGRV was chosen for the following reasons:

1. It is the largest system in the COMINT category.
2. It is delivered and operational.
3. It is a real-time embedded system.
4. It is a part of the Guardrail Common Sensor Family.

AQL was chosen for the following reasons:

1. It is the second largest system in the ELINT category.
2. Modifications to it are being made in Ada.
3. It is a real-time embedded system.
4. It is part of the Guardrail Common Sensor Family.

CHAALS was chosen for the following reasons:

1. It is the second largest system in the COMINT category.
2. Although not delivered, and thus not operational, its preliminary design is in Ada Program Design Language (PDL).
3. It is a real-time embedded system.
4. It is part of the Guardrail Common Sensor Family.

Trailblazer B was chosen for the following reasons:

1. It is the third largest system in the COMINT category.
2. It is a real-time embedded system.
3. The system is delivered and operational.

3.0 MAPPING SYSTEM CAPABILITIES TO ADA CONSTRUCTS

Each capability of COMINT/ELINT systems was mapped to the Ada constructs that would be used to implement that particular capability. The objective of this was to determine what Ada constructs would be used to implement the selected class of real-time embedded systems. The results were that all Ada constructs would be used in one of these systems. Also, no Ada construct could be determined to be more important than any other Ada construct because most constructs would be used throughout real-time systems. Because of this, the results of this step had no significance in the prioritization of RTE elements.

The system capabilities were mapped to both micro and macro constructs. For clarity, a discussion of system capabilities and the definition of a construct will be presented before the actual matrices.

3.1 SYSTEM CAPABILITIES

The system capabilities, i.e., intercept, direction finding, emitter location, analysis, and reporting, were found to be common throughout COMINT/ELINT systems. The way the systems specifically performed a particular capability might be different, but the overall objective was the same. Two systems that rely on other systems to perform one of the capabilities. For example, CHAALS relies on IGRV to perform its interception.

Because of the amount of effort required to perform an in-depth analysis to obtain base line Ada constructs, one COMINT/ELINT system was studied in detail. This analysis was performed by decomposing high-level system capabilities into low-level system capabilities and mapping these low-level capabilities to Ada constructs. Once the base line set of constructs was developed for one system, it was validated by comparing to the low-level capabilities of other systems in the same category to those of the system studied in detail. Four COMINT/ELINT systems were used to generate the Ada constructs. One system was used to establish the base

line, and the other three systems were used to validate the base line constructs.

3.2 ADA CONSTRUCT DEFINITION

While performing a preliminary review of the systems, it became obvious that 'construct' needed to be defined. Two definitions were appropriate: one for a micro construct and one for a macro construct. At the micro level, a construct was defined as an individual Ada statement. At the macro level, a construct was defined as a set of Ada statements that performs a well defined process. For this research, both micro and macro constructs were studied.

3.3 ADVANTAGES AND DISADVANTAGES OF MICRO AND MACRO CONSTRUCTS

The use of either macro or micro constructs has its respective advantages and disadvantages. The advantage of benchmarking micro constructs is that they are specific to each individual Ada statement, and therefore, each statement can be benchmarked. The disadvantage of benchmarking micro constructs is that interactions of individual statements when used for a particular application are ignored. Benchmarking only micro constructs is unrealistic compared to how Ada code is written. The advantage of benchmarking macro constructs is that they take into account the blending and interaction of Ada statements. Macro constructs are realistic to how Ada code is actually used. The disadvantage of benchmarking macro constructs is that they are only as good as the match between the benchmark run and the actual application code. The benchmark is some generic code used to carry out a particular process. If the actual application code varies from the generic code, the benchmark might not be valid.

3.4 SYSTEM CAPABILITIES VS. ADA CONSTRUCT MATRIX

Figure 3-1 presents the mapping of the system capabilities to the base line macro constructs. Figure 3-2 presents the mapping of system

capabilities to the base line micro constructs More information on how the Ada constructs were identified is provided in Appendix B.

System Capabilities Vs. Macro Construct Matrix

Capabilities	Macro Construct						
	MailBox	Queue	Event Flag	Semaphores	Stack Push Pop	Matrix Manipulation	Trigonometry functions
Intercept		0	0				
Direction Finding/ Emitter Location		0	0	0	0		
Analysis	0	0	0	0		0	0
Reporting			0		0		

Figure 3-1. System Capabilities Vs. Macro Construct Matrix.

System Capabilities Vs. Micro Construct Matrix

Micro Construct

Capabilities	Tasks	Selection Criteria	Delay Function	Clock Function	Procedure Call	function Call	Memory Allocation Deallocation	Exceptions	Priorities	Assignment Statements	Control Statements	Address Clause	Interrupt
Intercept	0						0					0	0
Direction Finding/ Emitter Location			0	0	0			0	0	0	0	0	0
Analysis	0	0	0	0	0	0	0	0	0	0	0	0	0
Reporting	0				0	0	0	0				0	0

Figure 3-2. System Capabilities Vs. Micro Construct Matrix.

4.0 MAPPING ADA CONSTRUCTS TO ADA RTE ELEMENTS

Each Ada construct was mapped to the Ada RTE elements that would be used to manage the implementation of that particular construct. A description of each of the RTE elements is provided in Appendix C [ARTEWG 1988]. The objective of this step was to determine which of the 11 RTE elements were not important for the class of real-time embedded systems being studied. The results were that every RTE element, except target housekeeping, had an Ada construct directly mapped to it; therefore, it was assumed that every RTE element was important except target housekeeping. The next step was to determine which of the remaining 10 RTE elements were the most important. A prioritization matrix (See Section 5.0) was developed. The implementation of this matrix, contradicted the earlier finding of this step that target housekeeping was not important. As a result of the matrix implementation, target housekeeping was deemed to be important; therefore it was considered in this study.

4.1 ADA CONSTRUCT VS. ADA RTE ELEMENTS MATRIX

The matrix that maps Ada macro constructs to the Ada RTE elements is shown in Figure 4-1. The matrix that maps Ada micro constructs to the Ada RTE elements is shown in Figure 4-2.

4.2 THE REASONING FOR MAPPING A PARTICULAR ADA CONSTRUCT TO A PARTICULAR RTE ELEMENT

A discussion of each identified Ada construct and why it is mapped to the particular RTE element is presented in the following subsections. One element, target housekeeping, had no Ada constructs mapped to it.

4.2.1 Dynamic Memory Management

The micro construct that maps to dynamic memory management and the Ada statement that allocates memory are indicated by the reserved word, NEW. For deallocation, Ada performs its own 'garbage collection'. Memory can be

Ada Macro Construct Vs. Ada Runtime Environment Elements

Taxonomy	Macro Construct						
	MailBox	Queue	Event Flag	Semaphores	Stack Push Pop	Matrix Manipulation	Trigonometry Functions
Dynamic Memory		0			0		
Processor Management	0			0			
Interrupt Management							
Time Management							
Exception Management							
Rendezvous Management	0			0			
Task Activities	0			0			
Task Termination	0			0			
I/O Management							
Commonly Called Sequences	0	0	0	0	0	0	0
Target Housekeeping							

Figure 4-1. Ada Macro Constructs Vs. Ada Runtime Environment Elements.

Ada Micro Construct Vs. Ada Runtime Environment Elements

Taxonomy	Micro Construct: Tasks	Selection Criteria	Delay Function	Clock Function	Procedure Call	Function Call	Memory Allocation Deallocation	Exceptions	Priorities	Assignment Statements	Control Statements	Address Clause	Interrupt
Dynamic Memory							0	0					
Processor Management	0								0				
Interrupt Management												0	0
Time Management		0	0	0									
Exception Management							0	0					
Rendezvous Management	0	0											
Task Activation	0												
Task Termination	0												
I/O Management												0	
Commonly Called Sequences					0	0				0	0		
Target Housekeeping													

Figure 4-2. Ada Micro Constructs Vs. Ada Runtime Environment Elements.

cleared deliberately by using unchecked deallocation. The dynamic memory function can also raise a storage error, if a request for storage cannot be fulfilled. The macro constructs that were mapped to the dynamic memory function are queues and stacks. Both of these constructs dynamically allocate or deallocate memory when they add or remove data from their structure.

4.2.2 Processor Management

The execution and scheduling of the micro construct, task, is closely related to the processor management function. The processor management function implements the assignment of physical processors to tasks that are logically executing. The micro construct, priority, is used to assist the processor management function in determining which task is to be assigned to the processor next. The macro constructs that were identified for processor management are the mailbox and the semaphore. Both of these constructs involve the use of tasks within their implementation.

4.2.3 Interrupt Management

The micro construct, interrupt, identified in Chapter 13 of the Reference Manual for the Ada Programming Language [ANSI/MIL-STD-1815A-1983], is used for the interrupt management function. Interrupt is used to react to asynchronous events. The address clause has been identified because it is used to access a particular hardware address to initiate an interrupt.

4.2.4 Time Management

The two micro constructs used in the time management function are the delay statement and the clock. Time management is the portion of the RTE that supports the predefined package, Calender. The time management function cooperates with the rendezvous management function to implement timed entry calls and selective waits with delay alternatives.

4.2.5 Exception Management

The micro construct, exception, was identified for the exception management function. Exception management implements the Ada semantics for raising exceptions and determines if there is a matching handler for a raised exception. The memory allocation/deallocation operation was also identified. It is responsible for initiating a storage error exception if a request to allocate memory cannot be performed.

4.2.6 Rendezvous Management Task Activation Task Termination

Rendezvous management, task activation, and task termination are concerned with the micro construct, task. Rendezvous management implements the semantics of the Ada rendezvous concept. Rendezvous management also concerns itself with the micro construct, selection criteria, which is used within a task. Task activation allows the dynamic creation of tasks. Task termination includes the set of rules for completion, termination, and abortion of tasks. The macro constructs mapped to these three elements are the mailbox and the semaphore. These were identified because they use tasks within their implementation. Tasks were chosen for their implementation because of the need for concurrent processing in real-time systems.

4.2.7 Input/Output (I/O) Management Function

The micro construct identified for the I/O management function is the address clause. The address clause is used for low-level I/O to communicate with physical devices.

4.2.8 Commonly Called Code Sequences

Commonly called code is the catch-all for the remaining micro constructs: procedure calls, function calls, assignment statements, and control statements. All of the macro constructs were mapped to the commonly called sequences because, if the constructs were included as a RTE predefined subroutines, they would be included under this function.

4.2.9 Target Housekeeping Functions

No micro or macro constructs were mapped directly to the target housekeeping functions.

5.0 STRUCTURE OF THE RTE ELEMENT PRIORITIZATION MATRIX

The initial plan for this research called for mapping system capabilities to the Ada constructs that would be used to implement them. These constructs would then be mapped to the Ada RTE elements necessary to support them. As Section 3.0 and 4.0 indicate, the diversity of Ada statements necessary to implement particular characteristics of the class of systems studied precluded using this approach to prioritize RTE elements. A new strategy was devised. The requirements for the selected systems were mapped to six basic features of real-time embedded systems. The 11 RTE elements were then prioritized based on their importance in implementing the six basic features. The key to this process was the use of a prioritization matrix. The remainder of this section details the development of this matrix and its implementation, i.e., prioritizing the RTE elements. The columns of the matrix are the six basic features of real-time embedded systems, and the rows are the 11 RTE elements.

5.1 THE SIX EMBEDDED COMPUTER SYSTEM (ECS) FEATURES

The Software Engineering Institute (SEI) determined that there are six basic features of an embedded real-time system: time control, concurrent control, I/O control, error handling, numeric computations, and internal representation [Weiderman 1987A]. The six features were developed by SEI from the definition, the general requirement, and the basic characteristics of embedded computer systems.

To implement the prioritization matrix, each of the features has a weight assigned to it. The weights represent the relative importance of an ECS feature with respect to the class of systems being studied. Each of the six features is given a weight, and the sum of the weights equal 100%. The 100% signifies an entire system within the class under study, and the separate weights indicate the importance of each feature to any system in that class of systems. The weights should not change as one moves from one system to another, provided one looks at systems in only one class. If the class of systems is changed, the weights will change.

Determining the weights for the COMINT/ELINT class involved two steps. The first was to understand which features were important and to begin to quantify their importance by studying the system requirements. For this study each requirement was mapped to the particular ECS feature to which it pertained. This step resulted with the majority of the requirements mapped to I/O control.

This first step gave an indication of which features were important and a number from which the feature could be assigned a weight; however, it did not take into account issues that effect the performance of a system. The primary concerns with respect to system performance were concurrent control and time control. The requirements may define the need for concurrency, but they do not represent the solution, which is the algorithm that is used to meet concurrency needs. Also, the requirements may define the time limits imposed on the system, but they do not reflect the stringency of those limits.

Step two was to adjust the weights by studying the requirements and determining their effect on the performance of the systems. Then, taking into account the results of steps 1 and 2, the weights were subjectively assigned to each ECS feature (see Table 5.1). A detailed discussion of the distinct characteristics of COMINT/ELINT systems that lead to the assignment of the final weights is presented in Appendix D.

Table 5-1
The Final Weights Assigned to the Features

Features	Weights
Concurrent Control	20%
Time Control	20%
I/O Control	25%
Error Handling	10%
Numeric Computation	10%
Internal Representation	15%

5.2 THE RTE ELEMENTS

The rows of the prioritization matrix are the 11 RTE elements that were obtained from the document "A Framework For Describing The Ada Runtime Environment" [ARTEWG 1988]. These RTE elements are the following:

Memory Management
Processor Management
Interrupt Management
Time Management
Exception Management
Rendezvous Management
Task Activation
Task Termination
I/O Management
Commonly Called Code Sequences
Target Housekeeping.

A detailed description of each RTE element can be found in Appendix C. The RTE elements make up the rows for the prioritization matrix. These elements are assigned rates. The rates are for quantifying the effect that an RTE element has on the performance of an ECS feature. Rating an element against an ECS feature is independent of the class of systems of interest.

A rating scale is used to rate an element. The scale shown below was used in this prioritization matrix. Following the scale, each classification is defined.

Intrinsic = 9
Supportive = 5
Extrinsic = 1

Intrinsic is defined as an RTE element that is foundational to the performance of a particular feature.

Supportive is defined as an RTE element that, although not intrinsic, has a role in the performance of a particular feature.

Extrinsic is defined as an RTE element that has at most a minor role in the performance of the particular feature.

Two documents were influential in the rating process: the ARTEWG document, "A Framework for Describing Ada Runtime Environments" [ARTEWG 1988] and the SEI document, "Ada for Embedded Systems: Issues and Questions" [Weiderman 1987A]. The rating process involved concentrating on one ECS feature to determine whether an RTE element was intrinsic to the performance of the feature. If it was, a '9' was entered into the square. If it was not, the RTE element was determined to be either supportive or extrinsic. A detailed discussion of each rating decision is presented in Appendix E.

5.3 APPLICATION OF THE RTE PRIORITIZATION MATRIX

After all the weights and rates had been determined the next step was to multiply the weights by the rates. This step integrated all of the components of the prioritization matrix: the ECS features, their relative importance to the class of systems (the weight), and the RTE elements' ratings.

The last step was to sum all the products in a given row. The result was a prioritized list of RTE elements. The element with the highest total for a row was the most critical element, and the element with the next highest was the next most critical, and so on.

5.4 THE MATRIX APPLIED TO THE COMINT/ELINT CLASS

Figure 5.1 presents the prioritization matrix for COMINT/ELINT systems.

ECS FEATURES

RTE ELEMENTS

	CONCURRENT CONTROL	TIME CONTROL	I/O CONTROL	ERROR HANDLING	NUMERIC COMPUTATIONS	INTERNAL REPRESENTATION	TOTAL
WEIGHTS	20	20	25	10	10	15	100
MEMORY MANAGEMENT	9 / 180	5 / 100	9 / 225	5 / 50	1 / 10	9 / 135	700
PROCESSOR MANAGEMENT	9 / 180	9 / 180	5 / 125	5 / 50	1 / 10	1 / 15	560
INTERRUPT MANAGEMENT	5 / 100	5 / 100	9 / 225	5 / 50	1 / 10	1 / 15	500
TIME MANAGEMENT	9 / 180	9 / 180	9 / 225	5 / 50	1 / 10	1 / 15	660
EXCEPTION MANAGEMENT	5 / 100	5 / 100	5 / 125	9 / 90	5 / 50	5 / 75	540
RENDEZVOUS MANAGEMENT	9 / 180	9 / 180	5 / 125	5 / 50	1 / 10	1 / 15	560
TASK ACTIVATION	9 / 180	5 / 100	1 / 25	5 / 50	1 / 10	1 / 15	380
TASK TERMINATION	9 / 180	5 / 100	1 / 25	5 / 50	1 / 10	1 / 15	380
I/O MANAGEMENT	5 / 100	9 / 180	9 / 225	5 / 50	1 / 10	5 / 75	640
CODE SEQUENCES	1 / 20	1 / 20	1 / 25	5 / 50	9 / 90	5 / 75	280
TARGET HOUSEKEEPING	1 / 20	5 / 100	1 / 25	5 / 50	5 / 50	9 / 135	380

RATING SCALE

INTRINSIC	9
SUPPORTIVE	5
EXTRINSIC	1

Figure 5-1. Prioritization Matrix for COMINT/ELINT Systems.

The following list is the prioritized list of RTE elements.

1. Memory management 700
2. Time management 660
3. I/O management 640
4. Processor management 560
5. Rendezvous management 560
6. Exception management 540
7. Interrupt management 500
8. Task Activation 380
9. Task Termination. 380
10. Target Housekeeping. 380
11. Commonly Called Code Sequences 280

This list of prioritized RTE elements is the driver for prioritizing groups of benchmarks.

6.0 PRIORITIZATION OF GROUPS OF BENCHMARKS

There is a very large number of available RTE benchmarks. Most of these benchmarks evaluate a single RTE element. Choosing which of the available benchmarks would be the most relevant for a particular application domain requires both prioritizing the RTE elements and mapping the existing benchmarks to the RTE elements. Each element, therefore, has a group of benchmarks mapped to it. Thus, it is these groups of benchmarks that have been prioritized, not the individual benchmarks.

Section 5.0 detailed the prioritization of RTE elements for COMINT/ELINT systems. This section provides the mapping of benchmarks to RTE elements.

6.1 MAPPING BENCHMARKS TO RTE ELEMENTS

The majority of the benchmarks listed here came from the document "Real-Time Performance Benchmarks for Ada" [Goel 1988]. The other benchmarks came from the Performance Issues Working Group [1988]. The Ada Compiler Evaluation Capability (ACEC) [Leavitt 1988] benchmarks were not included in this study because of disclosure restrictions. The format in which the benchmarks are presented and the numbers assigned to benchmarks were taken from their source. This has been done so that an individual can go back to the source document to obtain more information about a specific benchmark. The complete list of all the RTE elements and the prioritized groups of benchmarks is found in Appendix F.

For the purpose of mapping benchmarks to the RTE elements, two elements have been combined: task activation and task termination. This was done because the benchmarks that measure these two elements are similar, and the RTE elements have the same prioritization level.

TABLE 6-1 shows the priority order of the groups and the number of benchmarks.

TABLE 6-1

Priority of Benchmarks

RTE Elements		* of benchmarks
1.	Memory Management	16
2.	Time Management	4
3.	I/O Management	1
4.	Processor Management	5
5.	Rendezvous Management	22
6.	Exception Management	12
7.	Interrupt Management	6
8.	Task Activation/Termination	12
9.	Target Housekeeping	0
10.	Commonly Called Code Sequences	51

6.2 TYPES OF BENCHMARKS AND BENCHMARK DISTRIBUTION

To give the user a more thorough understanding of exactly what a benchmark measures, e.g., memory space or response time, the benchmarks were subdivided into types. It was determined that there were three types of benchmarks: timing benchmarks, storage benchmarks, and if-and-how benchmarks. The first two types measure the two critical resources of an embedded system, i.e., response time and memory space. The third type, if-and how benchmarks, address the need to determine how an RTE will respond given a set of conditions. The benchmarks reveal choices compiler vendors make when developing their RTE by determining if an RTE will implement a specific feature or how an RTE implements something. For example, a Processor Management benchmark, determine if user tasks are preemptive, will reveal how scheduling strategies were implemented by a particular vendor. If-and-how benchmarks will also be used to determine whether a particular feature is provided by a vendor, e.g., determine if unchecked deallocation is implemented.

6.2.1 Time Management Vs. Timing Benchmarks

Because of possible confusion, a distinction between time management and timing benchmarks needs to be made. Benchmarks that measure aspects of

time management pertain to Ada features that are time related, e.g., Measure CLOCK function overhead, Measure CLOCK resolution. Timing benchmarks relate to all those benchmarks that measure the length of time it takes for an event to occur, e.g., overhead time, time to store data, etc. Thus, if one is concerned with measuring the overall timing performance of an RTE, one should use timing benchmarks.

6.2.2 The Frequency of Commonly Called Code Sequence Benchmarks

Commonly Called Code Sequences had the most benchmarks, 51, mapped to it. ARTEWG describes this element as some what of a "catch-all" that includes runtime routines in the classical sense, e.g., multi-word arithmetic, block moves, and string operations. The types of benchmarks included in this group were procedure and function calls, addition, and anything that had code added (or removed, as with pragma PACK) by the compiler. Also included in this category were "code sequences" written by the user. While investigating the four systems, some code sequences resurfaced, e.g., matrix manipulation, trig functions, and message passing routines. All of these were put in the Commonly Called Code Sequences.

6.3 BENCHMARKS THAT NEED TO BE DEVELOPED

Once the benchmarks were mapped to the RTE elements, it became evident that some elements are not adequately addressed by benchmarks. It was determined that there is a need for additional benchmarks that evaluate some RTE elements. In some instances this need is because of an overall lack of benchmarks; in other instances, even with several benchmarks, others are needed. The elements in need of additional benchmarks are I/O Management, Processor Management, Target Housekeeping and Commonly Called Code Sequences.

Benchmarks need to be developed for I/O Management. During the course of the study, it was determined that for the COMINT/ELINT systems I/O is critical to system performance. With only one benchmark, it is difficult to determine an RTE's I/O performance.

More benchmarks need to be developed for Processor Management. Software developers must be able to determine courses of action taken by tasks in order that they might develop reliable programs. More if-and-how benchmarks would reveal to the software developer courses of actions implemented by the RTE.

Target Housekeeping is "associated with the actions starting up and terminating the execution environment of an Ada program" [ARTEWG 1988]. In one of the systems studied there is a requirement for the system to be up and running from a cold start in 10 minutes. This indicates that start up is critical, and thus benchmarks are needed to measure this element.

During the study of the four COMINT/ELINT systems, a number of 'algorithms' resurfaced. For example, each system's software solution frequently used matrix manipulations, trig functions, message transmission facilities, etc. Benchmarks for these 'algorithms' would be beneficial to the individuals selecting the compiler and RTE. These additional benchmarks would be mapped to the RTE element Commonly Called Code Sequences.

7.0 COMPOSITE BENCHMARK

The majority of benchmarks available either test a specific element of the RTE in isolation or exercise several elements in some unspecified combination. What is needed is a single benchmark that tests elements of an RTE while interacting in a manner that is consistent with their interaction during actual system operation. Such a benchmark would evaluate an Ada RTE by forcing the RTE to perform operations that would mirror the operations performed by the system to be developed.

7.1 PURPOSE OF A COMPOSITE BENCHMARK

A composite benchmark is a model of the capabilities of a particular class of systems. The purpose of a composite benchmark is to stress a computer and its RTE to evaluate their ability to perform the capabilities of a particular class of systems.

A composite benchmark allows a software developer to run one benchmark that will give him a general idea of whether a particular RTE can perform the capabilities of a given class of systems. A composite benchmark tests each capability individually and, more importantly, the interaction among the capabilities.

7.2 HOW TO DEVELOP A COMPOSITE BENCHMARK

Developing a composite benchmark description for a particular class of systems is not a simple task, but once developed the benchmark could be used to aid in the selection of an RTE for any system in the given class. The following three steps should be followed when developing composite benchmarks:

1. Identify the common capabilities of the particular class of systems by studying the requirements and functions of the systems within the class.

2. Define and analyze each capability. The description should include all functions common to the systems in the class. If a particular function

is common to several of the systems within the class, it should be included in the description because the function may be performed in a new system being developed.

3. Document the interactions and interfaces among the capabilities in a format that facilitates computer program code development. When writing the description, there needs to be continuous interaction between the writer and computer programmer to ensure that the composite benchmark will be accurate and understandable. The description writer must have an in-depth technical knowledge of the class of systems being studied.

7.3 COMPOSITE BENCHMARK FOR COMINT/ELINT SYSTEMS

A description for a preliminary composite benchmark was developed in this study for COMINT/ELINT systems. The goal has been to develop the idea and an approach for developing a composite benchmark. Because of this, the composite benchmark being developed is immature and needs to be addressed more directly in the future. The preliminary composite benchmark models the five capabilities of COMINT/ELINT systems: intercept, direction finding, emitter location, analysis, and reporting. This description was given to another company, TAMSCO, for code development.

It is assumed that the target audience of the composite benchmark description is familiar with COMINT/ELINT systems. Terms that may not be familiar to the target audience are defined.

7.4 COMPOSITE BENCHMARK DESCRIPTION FOR COMINT/ELINT SYSTEMS

The program performance specifications documentation and the program design specifications documentation for the four selected COMINT/ELINT systems (See Section 2.0) were studied to obtain the information used in the description.

7.4.1 The Intercept Capability

The benchmark must perform automatic acquisition of unknown signals. It will search frequency bands to find and catalog unknown signals. This involves two different search capabilities: general search (GS) and directed search (DS).

7.4.1.1 General Search (GS)

GS is a broad based sampling of frequency activity. It monitors a number of frequency bands for emitter activity and reports the occurrence of detected signals. This involves automatic environment mapping within selected frequency bands with associated geographic areas of interest and selected signal types. The benchmark will specify the frequency bands, exclusion of frequencies, and signal class/type.

A GS plan will be developed. It will include a set of data parameters: start frequency, stop frequency, frequency step size, receiver bandwidth size, and signal class/type. The plan also includes exclusion frequencies that are specified to inhibit the reporting of signal activity at specified frequencies. The benchmark will step through frequencies defined in the GS plan at a rate of at least 50 frequencies per second.

An activity table will be maintained. The GS activity table will contain entries for the most recent GS and manual direction finding (DF) that the system performed. The parameters stored in the activity table include the time of first and last intercept and a location estimate for each entry for which DFs have been taken. The emitter location can be determined automatically in response to GS activity in specified frequency bands.

7.4.1.2 Directed Search (DS)

DS is the automatic intercept of specific known signals. It involves automatic environment sampling at discrete frequencies, and it revisits

known emitters. The benchmark can specify a maximum of 20 frequencies for automatic activity detection.

The DS operation will monitor a list of individual frequencies specified in a DS plan and report newly active signals. The DS plan consists of the following: frequencies of interest, priorities associated with each frequency (normal, priority, monitor), the number of automatic DF requests to be made for each frequency, and simulate an operator's position being alerted when an intercept is detected by a DS. Some frequencies will be tagged for special handling such as increased sampling rate, prioritized audio monitoring, prioritized audio recording, and geographic screening. The DS plan contains at least 125 entries, including 20 priority DS entries and two monitor DS entries.

An activity table will be maintained for each specified frequency. The table includes an activity counter for each signal detected and the time of the last intercept.

Each specified frequency will have an associated priority assigned to it that is used to guarantee minimum revisit intervals. The software will determine the best DS frequency to be examined while taking into account the relative priorities of the frequencies. The software steps through the frequencies in the DS plan at a minimum rate of 50 entries per second. Monitor DS (highest priority) entries have a revisit time interval of no more than 0.1 second. Priority DS entries have a revisit time interval of no more than 0.5 second. Normal DS (lowest priority) entries have a revisit time interval as determined by the number of entries in the DS plan. Automatic signal analysis will be specified for a specific frequency and the results screened according to signal type/modulation.

7.4.2 Direction Finding (DF)

The benchmark will make various measurements that will provide an indication of the direction from which a frequency signal originated. The measurement process consists of several sequentially executed tasks that

conclude with the generation of a Line of Position (LOP). DF processing can be either automatically initiated or manually requested. The benchmark will accept input data from the DF related equipment and the magnetic field converter via an analog-to-digital converter.

An LOP consists of two pieces of data: a Line of Bearing (LOB) and the location of the receiving measurement equipment. An LOB is a line drawn from the measurement platform location at the angle (relative to north) that a signal arrived. When a LOB becomes referenced to a position (platform location at the time of the bearing), it becomes an LOP. The benchmark will compute and format an LOB message for a given DF request within 2.25 seconds of receipt of the request. The benchmark will store the LOB data from DSs, DFs, and manual DFs in the DF database segmented according to frequency.

The benchmark will schedule DF commands based on the following DF request priorities: manual DF, monitor DS, priority DS, normal DS, and GS. A local queue of pending DF requests is maintained. The benchmark will also allow voice and data activity related to the intercepted frequencies to be recorded.

7.4.3 Emitter Location

The benchmark will compute the location of an emitter signal. The benchmark will use the LOPs to compute the best estimate of the emitter location. Upon receipt of LOPs from a DF request initiated by DS or by the operator, the benchmark will attempt to associate the LOP set with an existing FIX location in a file. If an association is found, the LOP set is assigned to the corresponding FIX; otherwise, the benchmark attempts to generate an emitter location estimate. If a reasonable emitter location estimate cannot be determined, the LOP set remains unassigned. The benchmark will be capable of computing and displaying a FIX from five LOPs within 300 milliseconds. Provisions will be made for recognizing multiple emitters sharing common frequencies.

7.4.4 Analysis

The benchmark will allow automatic signal analysis, or it will simulate an operator manually requesting a signal analysis at the frequency he is monitoring with his intercept receiver. The following signal analysis data results are to be displayed: detected frequency, signal type or modulation, and audio classification. The signal classification section processes Signal Classification Tips (SCT) at a sustained rates of up to 20 per second without losing data. Then the benchmark will compare the results of signal classification to the acceptable list of types or modulations for the SCT.

7.4.5 Reporting

The benchmark will allow the simulation of an operator viewing DF data while generating a report. The data specification parameters allow the operator to view DF data associated with a single frequency or a frequency range, a specific time span within which the data was collected, or a specific geographical area. These data specification parameters are to be included in any combination.

The benchmark will generate reports semi-automatically for transmission by a reporting link. The software provides a means to facilitate the generation of reports and messages via prompts and displayed templates. The software formats the report generated into a form acceptable for transmission over the Reporting Data Link Subsystem (RDLS).

8.0 COMPILER AND RTE SELECTION PROCESS

The final step of this study was to determine how the benchmarks should be used when selecting an RTE. It was determined that choosing an RTE is a three-step process. The first step is to eliminate all RTEs that cannot perform beyond a minimum required threshold in each area critical to system performance. The second step is to begin with the set of RTEs that satisfy the minimum threshold requirements and select the small set of RTEs that performs best in the areas critical to system performance. The final step is to compare the costs, the vendor support provided, and any other mitigating circumstances for the final selection of an RTE or compiler. The first two steps involve the use of benchmarks.

The composite benchmark is to be used to test the minimum threshold of RTEs. This means the developer would only have to run one benchmark to eliminate all RTEs not suitable for his particular class of system. Then the other benchmarks would be used to test the remaining RTEs to see which RTEs perform the best in the areas critical to system performance. Because the RTE elements are prioritized, the critical areas and the benchmarks that measure those areas are known.

At this time the development of the composite benchmark is still in the preliminary phase. Until the composite benchmark matures, only the prioritized list of groups of benchmarks can be used to test RTEs.

9.0 SUMMARY AND CONCLUSIONS

The objective of this study was to provide software developers with guidance in the selection of a compiler and its RTE to ensure all the timing and storage requirements of real-time embedded systems, specifically COMINT/ELINT systems, are met.

To provide this guidance required the identification of Ada RTE features of interest for real-time systems. This involved two steps: first, the Ada runtime features that are important for real-time embedded systems were established; second, evaluation criteria to evaluate the features were determined. To determine the important Ada runtime features, the Ada RTE elements were prioritized. To evaluate the Ada runtime features, current benchmarks were prioritized, and new benchmarks were proposed.

Figure 9-1 provides the results of each step that was undertaken to prioritize benchmarks. In the first step, COMINT/ELINT systems were chosen to be studied, and the capabilities common to COMINT/ELINT systems were identified. Four systems were chosen for in-depth study because they were representative of all COMINT/ELINT systems. The second step identified what Ada constructs would be used to implement each capability. The third step identified which RTE element would be used to support the implementation of each identified Ada construct. The forth step prioritized the Ada RTE elements. The prioritization was done by applying a prioritization matrix to the RTE elements. It is recommended that when a developer prioritizes RTE elements and benchmarks for a particular class of systems, the developer begin by applying the prioritization matrix. The prioritization matrix was originally developed for COMINT/ELINT systems, but it can easily be modified to be used with other classes of systems. The final step prioritized benchmarks. The benchmarks mapped to a particular RTE element inherit the priority of that element. This study also presented the idea of a composite benchmark, which is one benchmark that will give the software developer a general idea whether a particular RTE can perform the capabilities of a

particular class of systems. A description for a composite benchmark was developed for COMINT/ELINT systems.

Benchmarks are used to identify RTEs and compilers that are best suited for a particular application domain by testing each candidate RTE to ensure all system requirements can be met. The results of this study, specifically the prioritization matrix, provide a process that a developer can use to prioritize benchmarks. If the critical elements of an RTE and compiler are not adequately evaluated, the selected RTE could be crippling for a real-time embedded system.

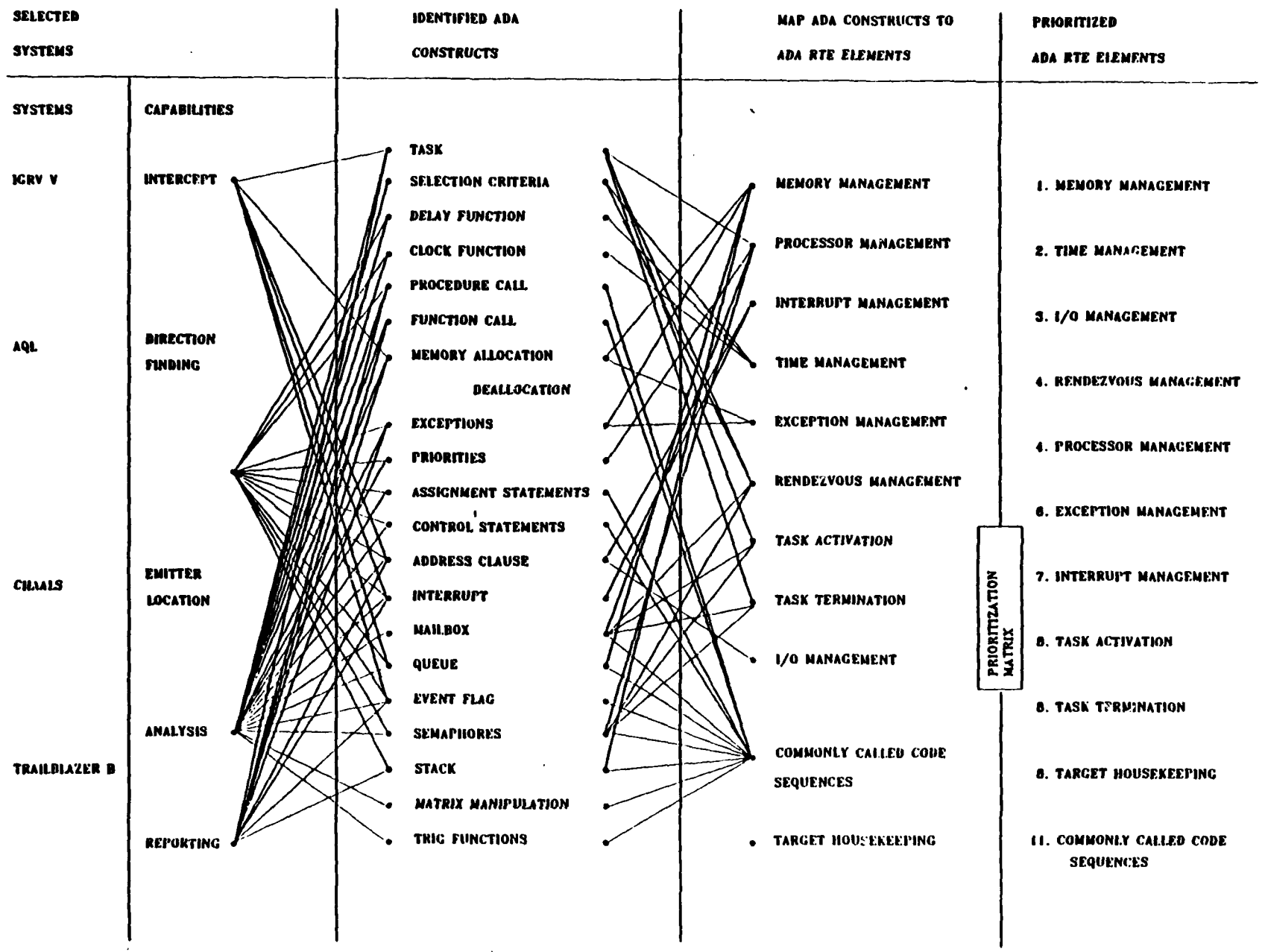

Figure 9-1. Mapping System Capabilities to Benchmarks.

PRIORITIZATION MATRIX

PRIORITIZED ADA RTE ELEMENTS	PRIORITIZED BENCHMARKS
1. MEMORY MANAGEMENT	
2. TIME MANAGEMENT	
3. I/O MANAGEMENT	
4. RENDEZVOUS MANAGEMENT	
4. PROCESSOR MANAGEMENT	
6. EXCEPTION MANAGEMENT	
7. INTERRUPT MANAGEMENT	
8. TASK ACTIVATION	
8. TASK TERMINATION	
8. TARGET HOUSEKEEPING	
11. COMMONLY CALLED CODE SEQUENCES	

Figure 9-1. Mapping System Capabilities to Benchmarks (Continued).

10.0 BIBLIOGRAPHY

ACM Ada Letters. 1987. International Workshop on Real-Time Ada Issues. Moretonhamsted, Devon, UK.

ARTEWG. 1988. A Framework for Describing Ada Runtime Environment. SIGAda.

Barnes, J.G.P. 1984. Programming in Ada. Addison-Wesley Publishing Company, Menlo Park, California.

Blackman, M. 1975. The Design of Real Time Applications. John Wiley & Sons Ltd.

Booch, Grady. 1983. Software Engineering in Ada. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California.

Eide, Arvid, R., et al. 1979. Engineering Fundamentals and Problem Solving. McGraw-Hill Book Company.

ESL Corporation. 1988. Trailblazer B SW Program Design Specification. Sunnyvale, California.

ESL Corporation. 1988. Trailblazer B SW Program Performance Specification. Sunnyvale, California.

ESL Corporation. 1985. IGR-V Computer Software/Firmware Document. Volume 1 through Volume 7, ESL Corporation, Sunnyvale, California.

Gehani, Narain. 1984. Ada Concurrent Programming. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Goel, Arvind Kumar. 1988. Real-Time Performance Benchmarks for Ada. Technical Management Services Corporation. Final Technical Report to Center for Software Engineering, CECOM, December 1988.

Habermann, A. Nico, Perry, Dewayne, E. 1983. Ada for Experienced Programmers. Addison-Wesley Publishing Company, Menlo Park, California.

IBM Corporation. 1986. Communication High Accuracy Airborne Location System (CHAALS). Program Design Specification, Omego, New York.

Jones, Robert E., Rosenberg, Mark. 1983. ARTE\DECO User's Guide. CECOM Project Control & Information Center.

LABTEK Corporation. 1988. Guidelines to Select, Use and Configure an Ada Runtime Environment, Final Technical Report to Center for Software Engineering, CECOM, December 1988.

LABTEK Corporation. 1987. Software Engineering Issues on Ada: Technology Insertion for Real-Time Embedded Systems, Final Technical Report to Center for Software Engineering, CECOM, July 1987.

Leavitt, T., Terrell, K. Ada Compiler Evaluation Capability (ACEC) Version Description Document. AFWAL-TR-88-1093. Boeing Military Airplane for Air Force Wright Aeronautical Laboratories. 1988.

Mellichamp, Duncan A. 1983. Real-Time Computing with Applications to Data Acquisition and Control. Van Nostrand Reinhold Company.

Performance Issues Working (PIWG). 1988. Systems Designers Software: Summary of PIWG Benchmark Results. Headquarters, Cambridge, Massachusetts.

Quirk, W. J. 1985. Verification and Validation of Real-Time Software. Springer-Verlag, Berlin Heidleberg New York Tokyo.

Ready, Jim, et al. 1988. Real-Time Applications with Ada. Tutorial at the Sixth National Conference on Ada.

Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983, Department of Defense, 17 February 1983.

Telos Federal Systems. 1986. CLCSE Weapon Systems Survey. Stewart Associates Incorporated, Red Bank New Jersey.

UTL Corporation. 1987. Advanced Quicklook Program Design Specification. Dallas, Texas.

UTL Corporation. 1987. Advanced Quicklook Program Performance Specification. Dallas, Texas.

Weiderman, Nelson, et al. 1987A. Ada for Embedded Systems: Issues and Questions. Software Engineering Institute (SEI). Carnegie Mellon University. Pittsburgh, PA.

Weiderman, Nelson, et al. 1987B. Annual Technical Report for Ada Embedded Systems Testbed Project. Software Engineering Institute (SEI). Carnegie Mellon University. Pittsburgh, PA.

APPENDIX A

REAL-TIME FUNCTION DESCRIPTIONS

This appendix contains the descriptions of the real-time functions introduced in Section 2.0. The functions described are those in TABLE 2-1. Also included is the list of functions from which the real-time functions were identified.

function descriptions

real-time characteristic: monitor

receiver - conversion of incoming electromagnetic waves into digital form

reception - action of receiving electromagnetic signals

target detection - finding the presence or existence of a moving target

real-time characteristic: analyze

direction finding - process of making measurements that indicate the direction from which a signal originated

analysis - interpretation and classification of signals

location - computing the location of a signal's origin

real-time characteristic: respond

transmitter - sending results of analysis to designated parties

countermeasure - after hostile missile detection, actions taken to counter its original intent

antenna controller - guiding the antenna to obtain the most efficient reception

APPENDIX B

HOW ADA CONSTRUCTS WERE IDENTIFIED

The following is the discussion on how Ada constructs were identified. For each system capability (intercept, direction finding, analysis, emitter location, and reporting) the system operations that perform particular capabilities are identified. Then, the Ada constructs that perform a particular operation are identified along with an example of how the construct is used.

Due to the amount of effort required for an in-depth analysis to obtain base line Ada constructs, one IEW COMINT/ELINT system was originally studied. These base line constructs were then validated by comparing them to other IEW COMINT/ELINT systems.

Because of the similarities between direction finding and emitter location, these two capabilities were combined.

C.1 Intercept

Intercept's major function is to determine signal presence.

The critical constructs used for interception are listed with explanations of how it would be used.

Micro Constructs

address clause	: used whenever an interrupt is used allocate needed
allocation	: memory for incoming data to tell that a message is
interrupt	: waiting to be sent and to indicate that a message is coming from another CPU
tasks	: used to continually poll the interface board

Macro Constructs

flags	: indicate the following: buffer in use, buffer is full, CPU is using another resource
queue	: when intercept is detected the data is put into a queue going to direction finding

C.2 Direction Finding and Emitter Location

Direction Finding (DF)

The major operations involved in DF are scheduling DF requests, removing requests that have not been processed in a set amount of time, and reporting the DF response to the system computer.

The critical constructs used in the DF were listed with examples of how they would be used.

Micro Constructs

clock	: for a timeout for a specific period of time
delay	: used to initiate the timeout
exception	: raised if audio correlation cannot be done because access is blocked
interrupt	: interrupt the audio correlator to send message to system computer
priority	: to establish priorities for direction finding requests

Macro Constructs

queue	: the DF output is stored in queues how the DF requests
stacks	: are stored determine if audio correlation is free
semaphore	:

DF Algorithm

The DF Algorithm is responsible for starting the data collection, cycling through the data collected, accumulating the data in the case of DF requests, and calculating the Line of Bearing (LOB).

The critical constructs used in the DF Algorithm will be listed with examples of how it would be used.

Micro Constructs

allocation	: after data is determined to be valid, memory is allocated to store the data

delay : used to allow an analog device to "settle" before taking baseline measurement
exceptions : a message is generated to indicate an error
interrupts : used to indicate the system has completed the current command

Macro Constructs

flag : indicate a process has occurred messages sent are
queue : queued so they can be read when ready

Navigation System

The Navigation System is responsible for reading navigational information from the Inertial Navigation System (INS). This is used to determine the emitter location.

The critical constructs used in the Navigation System were listed with examples of how they would be used.

Micro Constructs

control state : decodes the commands and calls the appropriate subroutine to execute the command
procedure : calling the subroutine moving date problem with
assignment : updating the system
exception :

Macro Construct

event flag : was identified to indicate that data has been stored in the data buffer from the incoming serial port

C.3 Analysis

System Administrators

The System Administrators serve as the controlling CPUs. They authorize the analysis CPUs to begin processing and control the interfaces between computers. The System Administrators control link handling, scheduling, directed and general search, and list handling.

The critical constructs used in the Administrators were listed with examples of how they would be used.

Micro Constructs

For the Ground Digital Administrator every micro construct was identified because it performs such a large variety of functions including scheduling, controlling and memory allocation. Several examples of the micro constructs are presented.

tasks	: used for continuous looping to check the response queue for messages received
clock	: requesting status information at regular intervals
delay	: reschedules itself using timeouts allocate memory for
memory alloc.	: received messages interrupting the system computer
interrupt	: for incoming direction finding data

Macro Constructs

mailbox	: used to send messages to the system computer queue
queue	: messages to be sent up the link to reserve the
semaphore	: output link queue to indicate a message has been
event flag	: received

Signal Classification and Recognition (SCAR) Analysis

The major functions involved in SCAR analysis include CPU system initialization, SCAR analysis control, SCAR calibration calculations, SCAR analysis calculations, SCAR discriminant calculation, and SCAR feature vector calculations.

The critical constructs used in the SCAR Analysis will be listed with examples of how they would be used.

Micro Constructs

assignment	: assign the results of mathematical calculations
interrupt	: SCAR CPU interrupted to receive message from the administrator

function : calling mathematical functions

Macro Construct

mailbox : to indirectly pass messages queue SCAR requests lock
queue : the database from being updated
semaphore :

SCAR Mathematical Analysis Functions

These are routines that perform mathematical functions necessary to accomplish SCAR analysis.

Two micro constructs were identified to perform the mathematical calculations: functions and the assignment statement.

Macro Construct

matrix manipulation : involves dividing, multiplying, adding and subtracting matrices of data

Trigonometry functions : solving SINE and COSINE functions

Analysis Library Routines

The library routines consist of functions needed by many different routines. The functions include the memory management routines, inter-CPU message passing, a random number generator, a queue flushing routine, the accountability number generator, a frequency offset adjuster, and directed search entry address calculations.

The critical constructs used in the Analysis Library Routines will be listed with examples of how they would be used.

Micro Constructs

interrupt	: interrupt to receive a message raised when message
exception	: is having trouble being passed

Macro Constructs

queue	: to queue messages sent down the link used to
mailbox	: indirectly send messages indicate the mailbox is in
semaphore	: use indicate a message has been read
event flag	:

C.4 Reporting

Uplink Multiplexer Software

The uplink multiplexer provides for communication. Memory space is allocated for databases and scratch pad memory. I/O ports are initialized, and the microprocessor instructions and memory (RAM and ROM) are verified.

The critical constructs used in the Uplink Multiplexer Software will be listed with examples of how they would be used.

Micro Constructs

dynamic memory	: create memory space for the database if memory cannot
exception	: be allocated to interrupt the Receiver Control Unit
interrupt	:

Macro Constructs

event flag	: to indicate the receipt of data how the data is stored
stack	: in memory

Intercom/Spectrum Display (IC/SD)

The IC/SD processor is responsible for controlling the Integrated Processing Facility Intercom System and for providing the spectrum display.

The critical constructs used in the IC/SD were listed with examples of how they would be used.

Micro Construct

tasks	: used in a polling loop waiting for activity
procedures	: calling the appropriate routine based on what was received from the polling loop
exception	: raised if there is a failure in passing data
functions	: functions called for testing

Macro Construct

event flag	: is used to indicate that data has been received.

APPENDIX C

ADA RUNTIME ENVIRONMENT DEFINITION

ARTEWG defines an Ada RTE as the set of all capabilities provided by three basic elements: predefined subroutines, abstract data structures, and code sequences.

Predefined Subroutines

The predefined subroutines are used by the compiler generated code to support features of the Ada language that the Ada implementor (vendor) has chosen not to directly represent in generated code. The set of predefined subroutines for a generated Ada program is called the Runtime System for that particular program. These predefined subroutines are chosen from the Runtime Libraries.

Abstract Data Structure

An abstract data structure is a grouping of related data items in memory. The items in a data structure can be processed individually, although some operations may be performed on the structure as a whole.

Code Sequences

Code sequences are commonly used, repetitious lines of code adopted for reliability, interoperability, and suppression of unnecessary implementation details. Code sequences are heavily affected by the selected definitions of predefined types and representations used for addressing objects. Related issues for code sequences include whether there is one or multiple areas for package (i.e., global) data, what mechanism for uplevel referencing of objects is (e.g., static link or display) and what the subprogram call sequences and parameter passing mechanism are determined to be.

The partitioning of functionalities between code sequences and runtime subroutines presents another set of issues that must be resolved in the runtime model. The decisions regarding this partitioning are influenced by the capabilities and limitations of the target configuration (e.g., how much of the tasking constructs are handled by inline code versus calls to routines). The best decision regarding allocation to either code or runtime routines is highly situational and must be based upon the particular target architecture and performance goals.

The runtime model resulting from the decisions described above defines the requirements of the design for the RTE components listed in the taxonomy. The following taxonomy describes a list of 11 functions that can be expected in the runtime libraries for Ada implementation.

Dynamic Memory

The dynamic memory management function is the part of the RTE that handles the allocation and deallocation of storage at runtime. If a request for storage cannot be fulfilled a storage error will be raised.

Processor Management

The processor management function implements the assignment of physical processors to tasks that are "logically executing." The processor management function is invoked by other components of the RTE to block and unblock tasks. It maintains a list of task priorities to determine which task should be assigned to a processor.

Interrupt Management

The interrupt management function implements the interrupts for asynchronous events in the underlying competing resource. It is not only responsible for initializing the interrupt mechanism in the underlying resource, but is also responsible for resetting the mechanism after the interrupt has occurred.

Time Management

The time management function is the portion of the RTE that supports the predefined package, Calender. This includes the support for the clock function and delay statement.

Exception Management

The exception management function implements the Ada semantics for exception handling. It determines whether there is a matching handler for the exception. If one is present it transfers control to the handler. If no matching handler is available it invokes the task termination function to terminate the task at hand or to terminate the main program. Both predefined and user-defined Ada exceptions may be raised.

Rendezvous Management

The rendezvous management function implements the semantics of the Ada rendezvous model. It monitors which tasks are blocked because they are waiting to rendezvous with other tasks; and it determines the exact circumstances of these wait states.

Task Activation

The Ada Language allows the dynamic creation of tasks. The task activation function is invoked by the creator of a new task in order to start the new tasks activation.

Task Termination

The task termination function contains the set of rules for the completion, termination and abortion of tasks.

I/O Management

This part of the Ada runtime environment supports input and output, including all of the functions that support predefined packages from Chapter 14 of the Ada Reference Manual.

Commonly Called Sequences

This category is a "catch all." It includes runtime routines in the classical sense, commonly called sequences of code.

Target Housekeeping Functions

Target Housekeeping functions are the parts of the Ada RTE that are responsible for starting up and terminating the execution environment of an Ada program.

APPENDIX D
WEIGHTING OF ECS FEATURES

I/O Control

COMINT/ELINT systems have strong dependence on input and output. The major aspect of COMINT/ELINT systems is the interception and monitoring of either electronic or communications signals. The systems need to enable and disable devices, handle device interrupts, and be able to move data to and from data registers. I/O control is the highest weighted criteria in the decision matrix because of the number of requirements mapped to it and because the COMINT/ELINT Systems most important capability, Intercept, heavily involves I/O. Intercept is the most important capability since if no signals are found and intercepted, the system is useless.

Another major part of COMINT/ELINT I/O is the communications between CPUs. All COMINT/ELINT Systems are comprised of multiple CPUs ranging from microprocessor chips to large main frames (Perkin Elmer). There is extensive I/O between the CPU's. For example, consider the following: Three microprocessors located on an airplane interact to perform interceptions, direction finding, and signal analysis. Then the intercepted signal and the analysis is sent to three microprocessors on the ground through one microprocessor dedicated for multiplex communication. On the ground the signal is further analyzed for location and characteristics. Then the signal is sent to a main frame computer for continued analysis by an operator and for generation of reports. For reporting the analyzed data to commands in the field, the information is sent back to the airplane for dissemination. This simple explanation of a complex process of I/O provides an idea of how extensively I/O between CPUs is used within COMINT/ELINT Systems.

Timing Control

Strict timing demands must be satisfied for COMINT/ELINT systems. The software requirements specify exactly what timing requirements must be met.

For example, in Trailblazer B the software must be capable of computing and displaying a FIX (location) from 5 lines of bearing within 300 milliseconds. If this and other timing constraints are not met, valuable data is lost or not analyzed. This valuable data cannot be recovered, but the system does not completely fail if a timing requirement is not met. Because of this, the timing requirements are strict but not critical. The timing control feature is the second highest feature.

Concurrent Control

One requirement for real-time embedded computer system is parallelism. COMINT/ELINT systems depend heavily on concurrent control. Concurrent processing must be used to meet the strict timing requirements needed to intercept, analyze, and disseminate COMINT/ELINT signals. Concurrent processing dramatically increases the speed at which data can be intercepted, analyzed and disseminated. Concurrent control allows for two capabilities, i.e., intercept and DF, to be performed concurrently. For example, one task can be dedicated to the interception and monitoring of signals on a particular frequency. This task will continually monitor the frequency without interruption, while another task can be dedicated to determine the direction from which the signal originated. If there were no concurrent processing the monitoring of a signal would have to be interrupted for a period of time while the direction was being determined. With this, valuable communication or electronic data could be lost. Another example is one task can be dynamically created to monitor each frequency from which a signal has been intercepted. This allows for the monitoring of multiple frequencies. When a frequency no longer has any activity, a task can be terminated.

Because concurrent control must be used to meet the strict timing requirements, the weighting of these to features are the same.

Internal Representation

COMINT/ELINT systems use low-level interfaces to communicate with hard wire devices (receiver control units) which perform the actual interception of the signals. This involves the conversion of the data signal from analog to digital. To store incoming data, efficient data representation in terms of the underlying computer architecture is needed. These involve the use of tightly packed data structures, dedicated memory locations, and special-purpose registers [Weiderman 1987A].

The highest weighted feature is I/O control. All the data that is intercepted must be stored in an efficient manner. This involves strict control of how and where the data is to be stored. This involves determining the amount of memory to be allocated for a particular data object and also the amount of memory to be allocated for a dynamically created task.

Internal Representation is weighted higher then error handling and numeric computations, because COMINT/ELINT systems rely heavily on I/O control.

Error Handling

Real-time embedded software must be reliable, where reliability is typically measured in terms of the system's availability, the mean time between failures, the meantime to repair and the frequency of failure. The normal approach developers have taken in order to meet reliability requirements is to design the Real-time system in such a manner that it can recover from its errors. Real-time software must be able to both detect and subsequently recover from errors [Weiderman 1987A].

Most COMINT/ELINT system have built in test equipment (BITE). BITE tests all the five capabilities each time before a system is activated for actual use. BITE can also be activated at any time during actual system

operation. Most errors should be detected before the system is in actual use so error handling is still important but not critical.

Numeric Computations

COMINT/ELINT systems rely on complex mathematical algorithms for analyzing, direction finding, and determining emitter location. The major importance is the time it takes to perform the algorithms and also the representation and implementation of the physical quantities (float point or fixed point). Numeric computations and error handling are important features, but for COMINT/ELINT systems they are the lowest weighted features.

APPENDIX E

RATING EACH RTE ELEMENT

The following is a discussion of each RTE element and its effect on the performance of each ECS feature. This section is divided by ECS feature and within each feature is a listing of each RTE element, its rating, and a brief discussion of how a particular rate was chosen.

Concurrent Control

RTE Element	Rating	Discussion
Memory Management	9	Memory management is intrinsic because of the need to store data during context switching. Also, there may be a need to dynamically create tasks in real-time embedded systems.
Processor Management	9	Processor management is intrinsic because it implements the assignment of physical processors to task that are logically executing when running parallel operations.
Interrupt Management	5	Interrupt management is a supportive because if the address clause for task entries are implemented, the interrupt management element utilizes the rendezvous management element to realize interrupt rendezvous [ARTWG87]. It is not intrinsic because interrupt rendezvous do not always have to be used.
Time Management	9	It is intrinsic because of the extensive use of time entry calls (delay statement) in real-time embedded systems.
Exception Management	5	A fault can be raised anytime during system execution. It is highly recommended but not mandatory that some type of exception management be used.
Rendezvous Management	9	The Rendezvous of tasks is intrinsic for concurrent control.
Task Activation	9	Task Activation is intrinsic for concurrent control.
Task Termination	9	Task Termination is intrinsic for concurrent control.
I/O Management	5	I/O Management is supportive because tasks can, but do not have to, be used during input and output.
Commonly Called Code Sequences (CCCS)	1	CCCS plays a minor role in concurrent control.
Target Housekeeping	1	Target Housekeeping plays a minor role in concurrent control.

Time Control

RTE Elements	Rating	Discussion
Memory Management	5	The time it takes to dynamically create variables or tasks may or may not be time critical.
Processor Management	9	The time in which a task is given sole use of the processor (in a uniprocessor system) is critical.
Interrupt Management	5	Interrupts from hardware timers may need to be passed on to the time management element to determine the length of the interrupt. This time period may or may not be time critical.
Time Management	9	Time Management is intrinsic because of the use of the package calendar and the delay statement in meeting timing constraints.
Exception Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Rendezvous Management	9	Time overhead to perform a rendezvous must be considered when trying to meet strict timing constraints.
Task Termination	5	The time it takes to terminate a task may or may not be critical. Task termination plays a role in time control, but it is not inherent.
Task Activation	5	The time it takes to activate a task may or may not be critical. Task activation plays a role in time control, but it is not inherent.
I/O Management	9	I/O in real-time embedded systems is subject to strict timing constraints. I/O management is intrinsic to time control.
CCCS	1	CCCS plays a minor role in time control.
Target Housekeeping	5	The time it takes to start up and terminate a computer system may be important for some real-time embedded systems.

I/O Control

RTE Elements	Rating	Discussion
Memory Management	9	During the input or output of data, memory is always being allocated or deallocated.
Processor Management	5	Ada tasks can be used to monitor asynchronous input. If they are, the processor management element will play a role.
Interrupt Management	9	Interrupt management is intrinsic because low-level asynchronous I/O operations to and from hardware devices are interrupt driven.
Exception Management	5	A fault can be raised anytime during system execution. It is highly recommended but not mandatory that some type of exception management be used.
Rendezvous Management	5	Ada tasks can be used to monitor asynchronous input. If they are, the rendezvous management element will play a role.
Task Activation	1	Task activation plays a minor role in I/O management.
Task Termination	1	Task Termination plays a minor role for I/O management.
I/O Management	9	The I/O Management element is intrinsic for I/O control.
CCCS	1	CCCS plays a minor role in I/O management.
Target Housekeeping	1	Target Housekeeping plays a minor role in I/O control.

Error Handling

RTE Element	Rating	Discussion
Memory Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Processor Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Interrupt Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Time Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Exception Management	9	Exception management is intrinsic to the performance of error handling in a embedded computer system.
Rendezvous Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Task Activation	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Task Termination	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
I/O Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.

CCCS	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Target Housekeeping	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.

Numeric Computations

RTE Elements	Rating	Discussion
Memory Management	1	Memory management plays a minor role in numeric computations.
Processor Management	1	Processor management plays a minor little role in numeric computations.
Interrupt Management	1	Interrupt management plays a minor role in numeric computations.
Time Management	1	Time management plays a minor role in numeric computations.
Exception Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Rendezvous Management	1	Rendezvous management plays a minor role in numeric computations.
Task Activation	1	Task activation plays a minor role in numeric computations.
Task Termination	1	Task Termination plays a minor role in numeric computations.
I/O Management	1	I/O Management plays a minor role in numeric computations.
CCCS	9	CCCS is intrinsic, because it includes runtime routine for multi-word arithmetic functions.
Target Housekeeping	5	Target housekeeping plays a role in numeric computations, because initial values of variable can be done during system initialization.

Internal Representation

RTE Elements	Rating	Discussion
Memory Management	9	Embedded computer systems must have strong control over dynamic storage and how variables and tasks are represented in storage.
Processor Management	1	Processor management plays a minor role in internal representation
Interrupt Management	1	Interrupt management plays a minor role in internal representation.
Time Management	1	Time management plays a minor role in internal representation.
Error Management	5	A fault can be raised anytime during system execution. It is highly recommended, but not mandatory, that some type of exception management be used.
Rendezvous Management	1	Rendezvous management plays a minor role in internal representation.
Task Activation	1	Task activation plays a minor role in internal representation.
Task Termination	1	Task termination plays a minor role in internal representation.
I/O Management	5	I/O management plays a role in internal representation, because how data is to be stored after it is input may be important in a real-time system in which storage is at a premium.
CCCS	5	CCCS plays a role in internal representation, because how the interim results of multi-word arithmetic problems are stored are important in a real-time embedded system in which storage is at a premium.
Target Housekeeping	9	Target Housekeeping is intrinsic for internal representation, because the declaration of variables (which determines how they are to be represented in storage) is done during system initialization.

APPENDIX F

PRIORITIZED BENCHMARK LIST

The following is the list of benchmarks grouped by the RTE element they measure. The order of the groups is a result of prioritizing the RTE elements. The benchmarks that measure the highest priority element are in the first group and the benchmarks that measure the second highest priority element are in the second group, and so on.

Each benchmark has a corresponding number. This number was taken directly from a benchmark's source which allows the reader to return to the source and obtain more information about a particular benchmark. Those benchmarks with identification numbers consisting of all digits and decimal points are from [GOEL 1988]. The others, whose identification numbers start with a letter and contain several zeros and end with a nonzero digit, are from [PIWG 1988].

MEMORY MANAGEMENT
MEMORY

4.1.1.1 Determine if task space is deallocated on return from a procedure (when a task that has been allocated via the new operator when that procedure terminates).

4.1.1.7 The attributes SIZE and STORAGE_SIZE provide information about storage assignments for task objects and types. These attributes can also be used to specify an exact size (amount of storage) to be associated with a task type. It is important to know how much storage a task object is allocated. Also how is runtime storage allocated for tasks? heap? stack?

4.3.1 Determine STORAGE_ERROR threshold.

4.3.2 Determine if Garbage collection is performed on the fly.

4.3.3 Determine if Garbage collection is performed on scope exit. In this test an access type to an array of 10000 integers is declared in a procedure called from the main program. This subprogram is called repeatedly and if storage is not being automatically deallocated upon scope exit, STORAGE_ERROR will again be raised. If garbage collection is implicitly called, no STORAGE_ERROR exception will be raised.

4.1.1.2 Determine if tasks that are allocated dynamically by the execution of a allocator do not have their space reclaimed upon termination when access type is declared in a library unit or outermost scope.

TIME

D000001 Dynamic array allocation, use and deallocation time. Dynamic array elaboration, 1000 integers in a procedure, get space and free it in the procedure on each call.

D000002 Dynamic array elaboration and initialization time allocation, initialization, use and deallocation 1000 integers initialized by others equal to 1.

D000003 Dynamic record allocation, and deallocation time elaborating, allocating and deallocating record containing a dynamic array of 1000 integers.

D000004 Dynamic record allocation, and deallocation time elaborating, initializing by (DYNAMIC_SIZE, (others equal to 1)) record containing a dynamic array of 1000 integers.

3.5.4.1 Measure time for allocating storage known at compile time.

3.5.4.2 Measure Time for Allocating Variable Amount of Storage

3.5.4.3 Memory Allocation via the New Allocator

3.5.4.4 Memory Allocation via the New Allocator when there are active tasks in the system

3.5.4.5 Determine the effect on time required for dynamic memory allocation when memory is continuously allocated without being freed.

IF-AND-HOW

4.3.4 Determine if Unchecked Deallocation is implemented.

TIME MANAGEMENT

TIME

3.4.2 Measure the actual delay time vs the specified delay time.

3.9.1.1 Measure CLOCK function overhead.

3.10.1.1 Measure the overhead associated with a call to and return from the "+" and "-" functions provided in the package CALENDAR.

IF-AND-HOW

3.9.1.2 Measure CLOCK resolution.

I/O MANAGEMENT

IF-AND-HOW

3.13.1.1 Determine if true asynchronous I/O is implemented. Benchmark Design: In the main procedure, three separate tasks are activated. Task 1 is the highest priority task, task 2 is medium priority, and task 3 is a low priority task. Task 1 makes a request from an I/O device, then task 2 makes a request to the same I/O device. Both task 1 and task 2 should be suspended and task 3 should be executing at this point.

PROCESSOR MANAGEMENT

TIME

4.1.3.3 Determine if a low priority task activation could result in a very long suspension of a high priority task.

IF-AND-HOW

3.4.1 Determine if user tasks are preemptive. Does a completed delay interrupt the currently executing task to allow the schedule to select the highest priority tasks.

4.2.1 Determine the method of sharing the processor within each priority to prevent starvation of any single task.

4.2.2 Does delay 0.0 simply return control to the calling task or causes scheduling of another task.

4.1.3.1 Determine priority of tasks (and of the main program) that have no defined priority.

RENDEZVOUS MANAGEMENT

TIME

3.3.2.2.1 Measure time for simple rendezvous

3.3.2.2.2 Measure time for simple rendezvous. More than one entry is called to measure rendezvous time. These entries can all be in a single task or single entries in multiple tasks.

3.3.2.2.3 Measure the affect on the time required for a simple rendezvous, where a procedure in the main program calls an entry in another task with no parameters as the number of accept alternatives in the selective wait increases. This benchmark is executed with the following scenarios:

3.3.2.2.4 Measure the affect of guards (on accept statements) on rendezvous time, where the main program calls an entry in another task (with no parameters) as the number of accept alternatives in the select statement increases. This benchmark is executed with the following scenarios:

3.3.2.2.5 Measure the time required for a complex rendezvous, where a procedure in the main program calls an entry in another task with different type, number and mode of the parameters.

3.3.2.2.6 Measure the affect on time required for a complex rendezvous, where the main program calls an entry in another task with different type, number and mode of the parameters as the number of accept alternatives in the select statement increase. The benchmark is executed with the following scenarios:

3.3.2.2.7 Measure the cost of using the terminate option in a select statement.

3.3.2.2.8 Measure the overhead due to a conditional entry call when a) the rendezvous is completed and b) the rendezvous is not completed

3.3.2.2.9 Measure overhead due to a timed entry call;

3.3.2.2.9 Measure the affect on time required for a complex rendezvous, where a procedure in the main program calls an entry as the number of activated tasks in the system increases.

3.3.2.2.10 Measure rendezvous latency.

T000001 Minimum rendezvous, entry call and return time. 1 task, 1 entry, task inside procedure, no select.

T000002 Tasking entry call and return time. 1 task active, 1 entry, task in a package, no select.

T000003 Tasking entry call and return time. 2 tasks active, 1 entry per task, task in a package, no select.

T000004 Tasking entry call and return time. 1 task active, 2 entries, task in a package, using select statement.

T000005 Tasking entry call and return time 10 tasks active, 1 entry per task, task in a package, no select.

T000006 Tasking entry call and return time. 1 task with 10 entries, task in a package, one select. Compare with T000005.

IF-AND-HOW

4.1.2.1 Determine algorithm used when choosing among branches of a selective wait statement. The implementation may make a) a random selection, b) select entry call that arrived first, c) select the first eligible accept alternative or d) select the task with the highest priority making the entry cal..

4.1.2.2 Determine the order of evaluation for guard conditions in a selective wait

4.1.2.3 Determine method used to select from delay alternatives of the same delay in a selective wait.

4.1.2.4 When are the expressions of an open delay alternative or the entry family index in an open accept alternative evaluated.

4.1.3.2 Determine priority of a rendezvous between two tasks without explicit priorities.

EXCEPTION MANAGEMENT

TIME

3.6.3.1	Measure a) timing overhead due to exceptions and b) exception response time when exception is handled in the block statement.
3.6.3.2	Measure a) timing overhead due to exceptions and b) exception response time when exception handled in the block statement while additional tasks are present in the system.
3.6.3.3	Measure Exception handling time when exception is raised and propagated one level below where it is handled.
3.6.3.4	Measure Exception handling time when exception is raised and propagated 3 levels below where it is handled.
3.6.3.5	Measure Exception handling time when exception is raised and propagated 4 levels below where it is handled.
3.6.3.6.1	Measure time to propagate TASKING_ERROR exception in the calling as well the called task.
3.6.3.6.2	Measure time to propagate and handle an exception when a child task has an error during its elaboration.
E000001	Time to raise and handle an exception. Exception defined locally and handled locally.
E000002	Time to raise and handle an exception. Exception is in a procedure in a package.
E000003	Time to raise and handle an exception. Exception is in a package, 4 deep.

IF-AND-HOW

4.1.1.5	If the allocation of a task object raises the exception STORAGE_ERROR, when is the exception raised? The LRM does not define when STORAGE_ERROR must be raised should a task object exceed the storage allocation of its creator or master. The exception must be no later than task activation: however an implementation may choose to raise it earlier.
4.4.1	Does an implementation raise NUMERIC_ERROR on an intermediate operation when the larger expression can be correctly computed?

INTERRUPT MANAGEMENT

TIME

3.8.3.1 Measure Interrupt Response Time

IF-AND-HOW

4.5.1.1 Determine if an interrupt entry call is implemented as a normal Ada entry call, a timed entry call, or a conditional entry call.

4.5.1.2 Determine if an interrupt is lost when an interrupt is being handled and another interrupt is received from the same device

4.5.1.4 Determine if an interrupt entry call invokes any scheduling decisions

4.5.1.5 Determine if an accept statement executes at the priority of the hardware interrupt, and if priority is reduced once a synchronization point is reached following the completion of accept statement.

4.5.1.6 Determine if entries can be called from application code.

TASK ACTIVATION

TASK TERMINATION

TIME

3.3.2.1.1	Measure task activation and termination time (without the new operator)
3.3.2.1.2	Measure activation/termination time for a) an array of tasks and b) task object declared as part of a record
3.3.2.1.3	Measure the time to activate a task created via the new allocator
3.3.2.1.4	Measure the time to activate and terminate a task object declared in the declarative part of a block as the number of existing active tasks keeps on increasing
3.3.2.1.5	Measure the time to activate and terminate a task created via the new allocator in a block as the number of existing active tasks keeps on increasing
C000001	Task create and terminate measurement with one task, no entries, when task is in a procedure using a task type in a package, no select, no loop.
C000002	Task create and terminate measurement, with one task, no entries, when task is in a procedure task defined and used in a procedure, no select, no loop.
C000003	Task create and terminate measurement, with one task, no entries when task is in declare block of main procedure, task is in the loop.

IF-AND-HOW

4.1.1.3	Determine the order of elaboration when several tasks are activated in parallel. When several tasks are activated in parallel, the order of their elaboration may affect program execution.
4.1.1.4	Determine if a task will continue execution following its activation but prior to the completion of activation of other tasks declared in the same declarative part. (See the Real-time Benchmarks paper for details)
4.1.1.5	What happens to tasks declared in a library package when the main program terminates? For some real-time embedded applications, it is desirable that such tasks do not terminate.

4.1.1.10.1 Determine order of evaluation of tasks named in an abort statement. An abort statement provides a convenient way to terminate a task hierarchy. When a task, T1, aborts a task, T2, the result T2'COMPLETED is true when evaluated by T1. Other tasks may not immediately detect that T2'COMPLETED is true. In real-time embedded systems, tasks may have to be aborted in a certain sequence. The semantics of the abort statement do not guarantee immediate completion of the named task. Completion must happen no later than when the task reaches a synchronization point.

4.1.1.10.2 Determine when an aborted task is complete from. When a task has been aborted, it may become completed at any point from the time the abort statement is executed until its next synchronization point. Depending on when an implementation actually causes the task to complete the results of an aborted may be different.

4.1.1.10.3 What are the results if a task is aborted while updating a variable? An implementation may defer completion of a task if it is aborted while updating a variable, and thus prevent a variable from being undefined. This may be crucial in the case of a common variable.

TARGET HOUSEKEEPING

COMMONLY CALLED CODE SEQUENCES

MEMORY

3.12.3.1 There are several test cases that are run with the pragma OPTIMIZE for option space. Determine the improvement in the size of the object code when this pragma is used.

TIME

3.7.1.1 This test measures the time to perform standard boolean operations (XOR, NOT, OR, AND) on arrays of booleans. The tests are performed on entire arrays.

3.7.1.2 This test measures the time to perform standard boolean operations (XOR, NOT, OR, AND) on arrays of booleans. The tests are performed on components of arrays.

3.7.1.3 This test measure the time to perform assignment and comparison operations on arrays of booleans.

3.7.1.4 This test measures the time to perform assignment and comparison operations on whole records.

3.7.2.1 Measure the time to do an unchecked conversion of one integer object to another.

3.7.2.2 Measure the time for UNCHECKED_CONVERSION to move a STRING object to another INTEGER object.

3.7.2.3 Measure the time to do an unchecked conversion of an array of 10 floating components into a record of 10 floating components.

3.7.3.1 Measure the time to store and extract bit fields using Boolean and Integer record components. 12 accesses, 5 stores, 1 record copy.

3.7.3.2 Measure the time to storage and extract bit fields that are defined by nested representation clauses using packed arrays of Boolean and Integer record components.

3.7.3.3 Measure the time to perform a change of representation from one record representation to another.

3.7.3.4 Measure the time to perform a change of representation from a packed array to an unpacked array.

3.7.3.5 Measure the time to perform POS, SUCC, and PRED operations on enumeration type with representation clause specification.

3.10.1.2 Determine the time to convert integer to Float using Float (I) and vice-versa using Integer (F).

3.10.2.1 Determine time required for float matrix multiplication.

3.10.2.2 Measure the time for a function that computes the inner (scaler) product of two values of type Vector where Vector is the following: type Real is digits...; type Vector is array (Integer range <>) of Real;

3.11.2.1.1 Measure the overhead and procedure call latency involved in entering and exiting a subprogram.

3.11.2.2.1 Repeat benchmarks from above with pragma INLINE for the called procedure.

3.11.2.3.1 Repeat benchmarks from above with the called subprogram being part of another package.

3.11.2.4.1 In the tests for inter- and intra-package calls, the subprograms are part of generic packages that are instantiated.

3.12.1.1 Determine improvement in execution speed when pragma Suppress is used for the following checks:

5.1.1.1 Measure time for a simple producer-consumer type transaction when the main procedure calls a consumer task.

5.1.1.2 Measure time for a producer-consumer type transaction when the consumer uses a selective wait. In this test the main task calls a consumer task that consumes more than one type of item.

5.1.1.3 Measure time for a producer-consumer type transaction when a producer task calls a consumer task.

5.1.2.1 In this benchmark, the producer task communicates with the consumer task indirectly through a bounded buffer.

5.1.3.1 In this benchmark, a producer task communicates with a consumer task indirectly through a bounded buffer with a transporter between the buffer and consumer.

5.1.4.1 In this benchmark, a producer task communicates with a consumer task indirectly through a bounded buffer with a transporter between the buffer and the producer as well as between the buffer and the consumer.

5.1.5.1 In this benchmark, a producer task communicates with a consumer via the relay. In terms of the task communication model, this resembles the producer-buffer-transporter-consumer paradigm, but

in terms of performance it should resemble the producer-buffer-consumer paradigm.

3.12.2.1 Determine the overhead due to Pragma SHARED when two tasks access a packed array of boolean shared variable.

3.12.2.2 Determine the rendezvous time when shared variable is updated during the rendezvous.

3.12.3.1 There are several test cases that are run with the pragma OPTIMIZE for option time. Determine the improvement in execution time of the object code when this pragma is used.

F000001 Time to set a boolean flag using logical equation. A local and a global integer are compared. Compare this test with F000002.

F000002 Time to set a boolean flag using an "if" test. A local and a global integer are compared. Compare this test with F000001.

L000001 Simple "for" loop time for I in 1..100 loop time is reported for once through loop.

L000002 Simple "while" loop time while I less than or equal to 100 loop time is reported for once through loop.

L000003 Simple "exit" loop time loop I:=I + 1; exit when I greater than 100; end loop; time is reported for once through loop.

P000001 Procedure call and return time. Procedure is local, no parameters.

P000002 Procedure call and return time. Procedure is local, no parameters, when procedure not inlineable.

P000003 Procedure call and return time. Procedure is in a separately compiled package. Compare to P000002.

P000004 Procedure call and return time. Procedure is in a separately compiled package. Pragma Inline used. Compare to P000001.

P000005 Procedure call and return time. Procedure is in a separately compiled package. One parameter, in INTEGER.

P000006 Procedure call and return time. Procedure is in a separately compiled package. One parameter, on INTEGER.

P000007 Procedure call and return time. Procedure is in a separately compiled package. One parameter, in out INTEGER.

P000010 Procedure call and return time. Compare to P000005. 10 parameters, in INTEGER.

P000011 Procedure call and return time. Compare to P000005 and P000010. 20 parameters, in INTEGER.

P000012 Procedure call and return time. Compare to P000010 (discrete vs. composite parameters). 10 parameters, in MY_RECORD, a three component record.

P000013 Procedure call and return time. 20 composite "in" parameters. The package body is compiled after the spec is used.

IF-AND-HOW

3.12.4.1 Determine if pragma CONTROLLED has any affect for a access type object.

4.6.1 Determine order in which actual parameters to a subprogram are evaluated?

4.6.2 Determine order in which parameters of modes out and in out are copied back at the completion of a subprogram call.

APPENDIX G

GLOSSARY

These definitions are being presented to facilitate the understanding of this report.

Ada Runtime Environment - A set of all capabilities provided by three basic elements: predefined subroutines, abstract data structures, and code sequences [ARTEWG 1988]

Analysis - the interpretation and classification of signals, also the portion of software that determines the next course of action based on the data collected

Direction Finding - process of making various measurements that will provide an indication of the direction from which a signal originated [ESL Corporation 1985]

Emitter Location - the computed location of a target [ESL Corporation 1985]

Intercept - determining signal presence and recording or monitoring it [ESL Corporation 1985]

Macro Construct - set of Ada statements that perform a well-defined process

Micro Construct - individual Ada statement

Real-Time - software that constantly monitors, analyzes, and responds to external physical events in a time-critical fashion [Mellichamp 1983]

Reporting - disseminating analyzed data [ESL Corporation 1985]

APPENDIX H

GLOSSARY OF ACRONYMS

AD - Air Defense

ANSI - American National Standards Institute

AQL - Advanced Quick Look

BFA - Battlefield Functional Area

CECOM - Communications and Electronics Command

CHAALS - Communication High Accuracy Airborne Location System

COMINT - Communications Intelligence

DoD - Department of Defense

DF - direction finding

DS - directed search

ECS - Embedded Computer System

ELINT - Electronics Intelligence

FS - Fire Support

GS - general search

IC/SD - Intercom/Spectrum Display

IEW - Intelligence Electronic Warfare

IGRV - Improved Guardrail V

I/O - Input/Output

LOP - Line of Position

LRM - Language Reference Manual

MC - Maneuver Control

PDL - Program Design Language

RDLS - Reporting Data Link Subsystem

RTE - Runtime Environment

SCAR - Signal Classification Recognition

SCT - Signal Classification Tips

SEI - Software Engineering Institute